Number: 11.01

HOST MAINFRAME PROGRAM PLANNING AND DESIGN

August 2001

TABLE OF CONTENTS

1 INTRODUCTION

Standardized program specifications should provide programmers with uniform specifications and information. This document will provide guidance for analysts who must develop program specifications and for programmers who receive program specifications.

1.1 Purpose

This document provides a common basis for the planning, development, and designing of standardized application programs. It is intended to provide guidance for data processing personnel who must develop applications or work with programming specifications.

1.2 Scope

The scope of this document is limited to the identification of the resources and forms that should be considered during the development of program specifications, the supporting information that is required for the development of the specifications, and a general description of how to present the information required for program documentation. This document does not cover application systems analysis, design, testing, and implementation.

1.3 Applicability

This document represents the State of Hawaii's Executive Branch's standards and guidelines for the planning, development, and designing of application programs. This document is intended to be used by both State agencies with data processing personnel and contractors working for the State.

Because this document is intended to be used by application programmers during the planning, development, and designing of applications in different programming languages, words such as "should," "avoid," "minimize," "try," or "encouraged" were included in the text. These words are intended to provide guidance for people who are enhancing existing programs. But for any new program, these words will be interpreted in their absolute sense of "will use," "do not use," or "shall use."

1.4 Requests for Training

Any requests for training in structured analysis, design, program development, or any programming language or technique should be coordinated through the agency's dp coordinator.

1.5 References

Throughout this document, the phrase "Standards Document 07.05" will refer to the State's “ICSD Operations Documentation Guide for Production Jobs." The term "Production Services Branch" or “PSB” refers to ICSD's "Production Services Branch." And the term "Applications Branch" refers to ICSD's "Client Services (Applications Systems Development) Branches."

1.6 Suggestions and Comments

Submit suggestions or questions relating to this standard to:

Department of Accounting and General Services
Information and Communication Services Division
Planning and Project Management Office

Telephone: 586-1920

2 PROGRAM PLANNING

The systems analyst for an application system should provide the programmer with all of the supporting documents for the successful creation of an application program. One of the key documents is the system flow diagram that should show the position of the program in the system. It shows the required input data and the required output from the program. This system flow diagram should be carefully reviewed before starting any program to acquaint the programmer with the environment that the program will be required to operate in. This diagram shows all the planned programs, data files, outputs, interfaces to and from the programs in the system, and interfacing utilities or any other supporting software.

For each input file in the System Flow Diagram related to the program, review the detailed description of the input file characteristics, and review the detailed layouts for the records, transactions, parameters, and any control statements entered or generated by the program that are specified on the program specification forms.

2.1 Program Specification Forms

The State has many forms for the systems analysts to use in developing their detailed system's internal specifications. The SDM documentation for the SIS phase has a description of all these forms. Their selection is dictated by the unique criteria and situations of the system and the program. Forms that have generally been used for many systems at ICSD are:

• EDPD S1	Request for Services
• EDPD A-123	Program Specification
• SDM 170	Work Flow Schematic
• SDM 130	General Procedure Description
• EDPD T-121	Data Set Description Sheet
• SDM 302	Detail File Description
• SDM 311	File/Record Content
• SDM 312	Segment or Record Description
• SDM 400	General Output Description
• SDM 401	Output Content
• SDM 320	Record Layout
• SDM 103	HIPO Chart - I
• SDM 205	Display Layout (for On-Line Systems)
• GX20-1816	150/10/6 Print Chart (150 Columns)

2.2 Naming Conventions

All job name, step name, data set name, source module, load module, include module, and program names must conform to the established naming conventions found in the State's "Job Control Language Standards," standards document 11.02.

2.3 Page Size Specifications

A line-counting feature should be designed for report printing control. For COBOL programs not using the Report Writer, the C01, End-of-Page special name should be used. And for PL/I programs, the ENDPAGE conditional control block should be used.

2.4 Console Messages

The only text that a program may send to the operator's console will be the status of a program's execution. Any other text to be sent to the operator's console must be approved by the Production Services Supervisor.

2.5 Job Re-execution Capabilities

As much as possible, application job streams will be designed so that they can be resubmitted by the Production Services control clerk. If the job stream cannot be resubmitted at any time, the aborted job stream will be returned to the submitter. For jobs that can be restarted at a step after the first job step, restart points must be documented in the operation's documentation. Any recovery procedures to be followed before a job is resubmitted by the Production Services control clerk must be defined in the operation's documentation.

2.5.1 Automatic Rerun

Jobs that can be rerun at any time must not affect the system catalog. These jobs are typically edit, query, extract, or print only runs.

2.5.2 Restart

There are two ways to define a restart for a job:

a. RESTART parameter on the JOB statement to restart a job at a job step (which is the preferred method), and

b. Checkpoints to restart a job within a job step.

If printed output cannot be regenerated or is critical to the restart procedure, the generated output should be directed to a temporary disk storage data set and the report printed at the end of the job stream.

2.6 Direct Access Storage

Direct access storage is a necessary resource for the storage and retrieval of an organization's data. ICSD has a limited number of devices that must be shared by all agencies.

2.6.1 Requesting Direct Access Storage

The Systems Services Branch (SSB) will make available any required permanent or temporary (i.e. "Test") storage for applications on shared devices. The project analyst must fill out and submit form ICSD T-121, "Data Set Description Sheet," to request direct access storage areas at ICSD. This form must be submitted for all disk storage requirements.

2.6.2 Permanent Data Sets

All permanent or production applications will have their data stored on the shared devices managed by the SSB. The maintenance of permanent application data sets is the responsibility of the agency. Example: Running Backup Jobs.

2.6.3 Temporary data Sets

Most program development requiring data to be saved during the development process should allocate data sets on temporary storage devices by using the JCL parameter, UNIT=3390,VOL=SER=XNZ001. Should a program require large amounts of storage, for example in excess of four cylinders, the project system analyst must make arrangements with the SSB. Temporary data sets will be purged on a regularly scheduled basis.

2.7 Magnetic Tape

Magnetic tape storage may be used for the storage and retrieval of an organization's data or for the long-term storage of data. There are a limited number of drives for these tapes at ICSD that must be shared by all user agencies. Any job stream requesting the availability of more than three tape drives must have the prior approval of the Job Scheduler in the Production Services Branch. A job should not use more than five (5) tape drives.

2.7.1 Requesting Magnetic Tape

The project systems analyst must refer to standards document 07.05 for the appropriate form and procedures to request tapes for any tape-oriented system.

2.7.2 Magnetic Tape Job Submission Procedures

Any test job requesting tape mounts must use a special job class code. The appropriate code and required job statement coding may be found in the JCL standards document 11.02. Any test job requiring tape mounts must be submitted to the Production Services Branch in an appropriate pouch with an ICSD Job Route card.

2.7.3 Storing Reel/Cartridge Tapes

ICSD has very limited space for the storage of reel/cartridge tapes. Only test jobs that require frequent access to tapes may be stored in the Production Services area. The project manager should ask the Production Services Supervisor for space on these racks. The storage and security of the other tapes are the responsibility of the user agency.

2.8 Generation Data Groups

Generation data groups are highly recommended for application systems. They provide a very good mechanism for auditing, history, backup, and recovery for the system.

The procedures for establishing a generation data group may be found in the JCL standards document 11.02.

2.9 Using Operating System Procedures

The system developer communicates with the operating system, job scheduler, and programs by job control statements. The ICSD compilation, load, and execution "PROCs" are cataloged in data set EDPD.PROCLIBU. They should be used to execute special functions in either the batch or on-line environment. These "PROCs" are listed in the JCL standards document 11.02.

2.9.1 Catalogued Procedures

The default parameters for the cataloged procedures were chosen to provide effective controls for the job scheduler to regulate the execution of steps, to retrieve and determine the disposition of data allocating resources, and to communicate effectively with the operators and programmers.

2.9.1.1 COBOL/MVS Procedures

ICSD has established cataloged procedures to be used for applications in a COBOL/MVS environment. There are procedures to support initial syntax checking; logical checking; and procedures to support the CICS environment.

The names of these procedures and their functions are presented in the "COBOL/MVS Standards and Conventions," standards document 11.10.

The procedures use Computer Associates CA-OPTIMIZER. They should be used to debug the logic of the COBOL application program.

The names of these procedures and their functions are presented in the "COBOL CA-Optimizer Procedures," standards document 11.01.06.

2.9.1.2 COBOL with CICS Procedures

COBOL programs developed for interactive applications using IBM CICS may be processed via cataloged procedures.

The names of these procedures and their functions are presented in the "COBOL/VS Standards and Conventions," standards document 11.10.

2.9.1.3 PL/I Procedures

ICSD has cataloged procedures for applications development in a PL/I environment. These procedures are for the users of the IBM OS PL/I Level F and IBM OS PL/I Optimizing Compiler.

These versions of the IBM OS PL/I Compiler installed at ICSD are not supported by IBM and are available to users on an as-is basis.

The names of these procedures and their functions are presented in the "PL/I Standards and Conventions," standards document 11.16.

2.9.2 Instream Procedures

Instream procedures are used like cataloged procedures. The difference is that cataloged procedures are members of a partitioned data set, and the statements for the in-stream procedure are placed in the job input stream after the JOB statement. In-stream procedure specifications should be used to test proposed production job streams. They may also be used for limited-use utility procedures. These in-stream procedures may be cataloged in the partitioned data set established for an agency or the "EDPD.PROCLIBA" at a later time for production runs. Any control statements for these procedures may be cataloged in the partitioned data set established for an agency or the "EDPD.PARMLIB" at a later time for production runs. The JCL standards document 11.02 has a more in-depth discussion on the topic "Using In-stream and Cataloged Procedures."

3 PROGRAM DESIGN

The program design provides for the visualization of all the pertinent logic identifying functions and information concerning the particular application. All structures, operations, and processes identified and described in the design should be incorporated in the program.

The application program's identifying name should be an eight character variable that conforms to the naming conventions described in the "Job Control Language Standards," standards document 11.02.

The program should be self-documenting, readable, and easy to modify or maintain by program maintenance personnel. To accomplish this, program labels, paragraphs, or variables must be consistent in spelling and meaningful enough to document the program's purpose, function, and logic.

3.1 Program Structure

The program's logical design will follow structured programming techniques. Structured programming is a style of programming in which the structure of a program is made as clear as possible by three control logic structures:

- Sequences
- Selections
- Iterations

These control logic structures may be combined to produce programs that handle information-processing tasks of varying complexities.

3.2 Program Function Structure Chart

The hierarchical flow of the processes that a structured program should accomplish must be drawn so that the data flow through the job functions to be executed can be traced in sequence, from top to bottom, without "skipping around." It is easier to comprehend a function when all the statements that may influence its action are physically close by. Top-down readability is one consequence that should be accomplished when these functional modules are developed by using only the previously defined control logic structures. An example of a hierarchical structured chart is shown in Figure 1.

Top-down program development is strongly recommended. This involves sketching a general flow of procedures, then iteratively write and test the hierarchical-level groups of functions. This gives the critical top segments the most testing and provides earlier warning of problems that may occur when all of the segments are integrated into the program. The inputs and outputs are listed and the processing related to them are specified. The hierarchical chart shows the functions and subfunctions and their interrelationships.

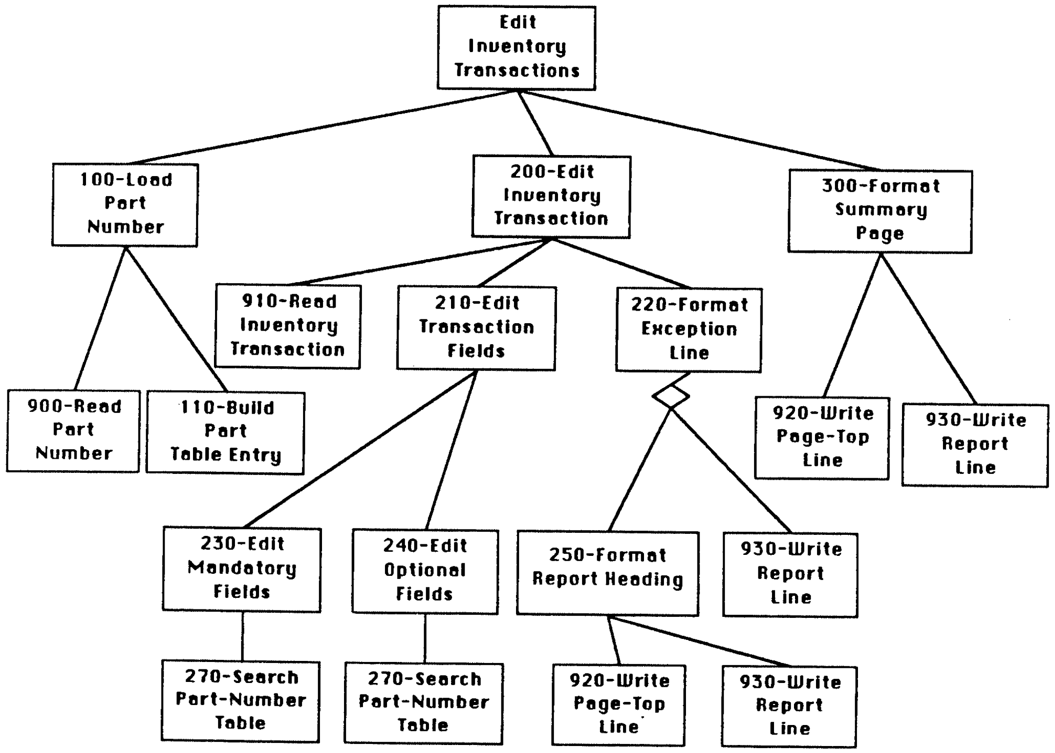

Edit Inventory Transactions
100-Load Part Number
200-Edit Inventory Transaction
300-Format Summary Page
910-Read Inventory Transaction
210-Edit Transaction Fields
220-Format Exception Line
900-Read Part Number
110-Build Part Table Entry
920-Write Page-Top Line
930-Write Report Line
230-Edit Mandatory Fields
240-Edit Optional Fields
250-Format Report Heading
930-Write Report Line
270-Search Part-Number Table
270-Search Part-Number Table
920-Write Page-Top Line
930-Write Report Line

Figure 1. Hierarchical structured chart.

3.3 Structured Programming

As stated in section 3.1, all programs must be designed and written to use only the control logic structures of sequence, selection, and iteration.

The program's logic and organization must be developed to satisfy two basic requirements. The first is that a structured program must be divided into logical segments, modules, or procedures. Each segment must have exactly one entry point and exactly one exit point for program control. The second criteria are that there must be logical paths from the entry point to the exit point that lead through every control logic construct of the program.

The three control logic constructs are defined as follows:

a. Sequences are statements or procedures that are executed in the order in which they appear in the program. The boxes in the following figure are labeled "A" and "B." These are function boxes that may contain anything from a single statement up to complete processing modules.

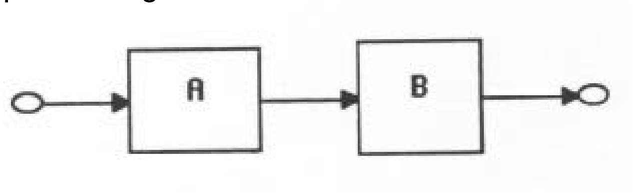


Figure 2. The sequence construct.

b. Selection is a group of statements that allows for a choice between two actions based on a predicate. In Figure 3, "P" is the predicate or conditional statement. This is the IF-THEN-ELSE structure.

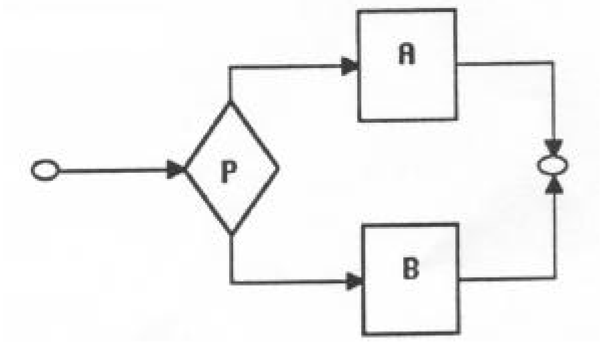


Figure 3.The selection construct.

c. The iteration is a control mechanism that is used for the repeated execution of code while a condition is true (also called loop control). As shown in the following figure, "P" is the predicate or conditional in a decision diamond and "A" is the controlled code in a function box.

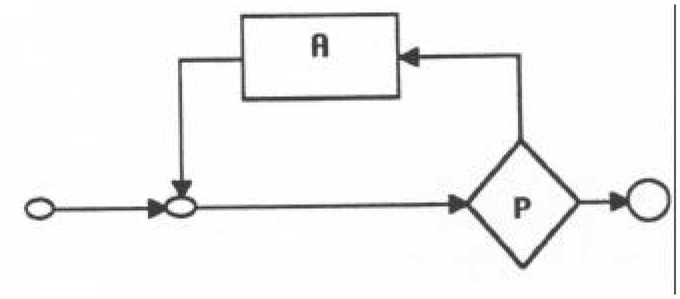


Figure 4. The iteration construct; (Do While Condition).

Any of the three basic structures may be substituted anywhere a function box appears and still be a proper program. Flowcharts of arbitrary complexity can be built up in this way.

The ability to substitute control logic structures for functions is called the nesting of structures. This is illustrated below.

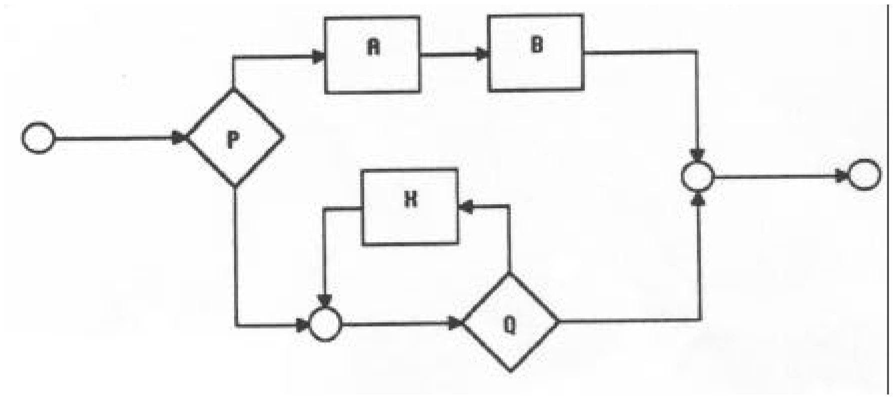


Figure 5. Nesting the basic constructs.

3.3.1 Iteration Variation

The basic iteration structure previously described is the DOWHILE structure but closely related to it is the DOUNTIL structure. The difference between the DOWHILE and DOUNTIL structures is that the DOWHILE predicate is tested before executing the function; but the predicate in the DOUNTIL is tested after executing the function. The DOUNTIL function will always execute at least once.

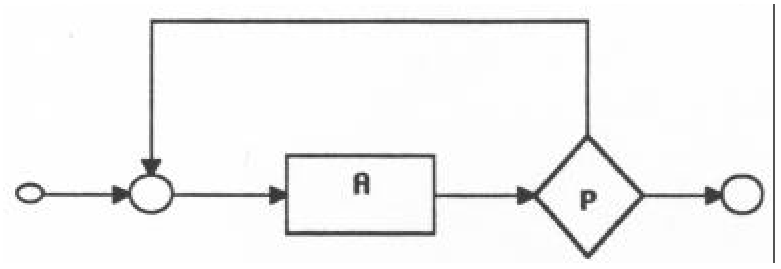


Figure 6. The loop variation. The DOUNTIL construct.

3.3.2 The CASE Selection Structures

A CASE structure is a special type of selection. It is used to express a mutually exclusive multiway branch. The value of a control variable will determine which one of several routines will be executed. These mutually exclusive branches could be implemented with multiple IF-THEN-ELSE statements. But the structure to be used in PL/I is the SELECT-WHEN-OTHERWISE.

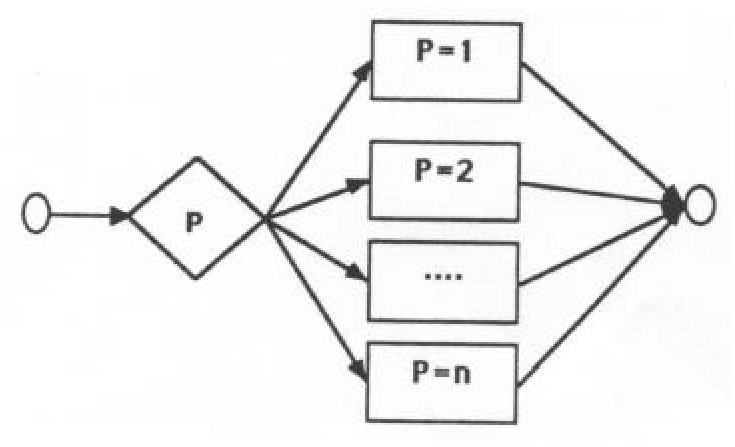


Figure 7. The selection variation. The CASE structure.

3.3.3 "GO TO" Statements and Labels

A well-structured program can be read in sequence without "skipping around." "GO TO" is not a standard control logic structure.

No special effort is required to eliminate "GO TOs." No extra effort is required to "avoid" them. "GO TOs" will not occur when the standard control logic structures are used. There are uncommon situations where the use of GO TOs may improve readability, such as exception conditions to abort the processing of a function.

Do not use varying label variables, which may cause a segment or procedure's label name to change.

3.3.4 Segmentation

Easy program readability minimizes the turning of many pages to understand how something works. A program segment should be kept to a page of code. Programs should be developed in logical functions (segments) such as initialization, house- keeping, data manipulation, main-line processes and sub-processes, I/O, and summarization. The selected label name for these segments will have numerical prefixes so that the external reference list generated by the compilers to list the label names in a structured sequence would document the logical flow of the segments. This range of prefixes has been standardized for COBOL and can be found in the "COBOL/VS Standards," document 11.10.

The characteristics of good program segmentation are:

a. The program is divided into logical pieces that relate to each other in a functional, hierarchical manner. Segments at the top of the hierarchy should contain high-level control function statements.

b. A well-designed segment carries out only functions and processes that are closely related to each other.

c. An identified segment communicates with other segments in controlled ways. For example in PL/I, segments consist of procedures and the only communication between them would be through parameter lists.

3.3.5 Identification

Consistent indentation enhances readability so that the finished program exhibits, in a pictorial way, the relationships among statements indented by a consistent amount to show the scope of control.

In pseudo-code process statements and actual language code, the processing associated with IF-THEN-ELSE will be indented right of the IF-THEN-ELSE clause. The THEN-ELSE clauses will be alone on a line.

The code controlled by a group or a block of codes should be indented to visually isolate the group of statements to be processed within the group or block.

Paragraph or label names will be left justified on a line by themselves.

Consistent indentation makes it easier to understand, to verify, and to check the logic for correctness.

An example of logical nesting of the IF-THEN-ELSE with consistent indentation is shown below:

```
IF CONDITION_P IS TRUE
          THEN DO;
              B = A + B;
              IF CONDITION_Q IS TRUE
                  THEN
                      C = 12;
                  ELSE DO;
                      C = 36;
                      IF CONDITION_R IS TRUE
                          THEN
                              Y = X + Y;
                          ELSE
                              Z = X + Z;
                      END_IF;
                  END_ELSE;
              END_IF;
          END_THEN;
          ELSE
              A = A + B;
      END_IF;
```

3.3.6 Establishing Identification Guidelines

Each dp agency will establish a local convention and follow it consistently. Whether statements are indented four, three, or two positions, the rule must be set and followed or else the value of indentation will be lost.

3.4 Flowchart Logic

If flowcharts are used to express logic flow, they must include only the three basic structures of sequences, selections, and iterations. The flowchart symbol convention is the same as those defined on the IBM programmer flowchart template jacket.

3.5 Pseudo-code Logic

The program's logic should be developed in pseudo-code structured English. The specifications from the program narrative are defined in consistent verbs and phrases that are used to express the three structured programming constructs. The pseudo-code logic statements should be incorporated as documentation.

The initial design of each identified major function to accomplish the objectives of the program is defined and then subdivided into smaller tasks. The initial design does not include low-level, how-to detail statements. The program developer manages complexity by evolving the problem solution one level of detail at a time.

The basic idea is to begin with a top-level logic, attach functional processes with a little detail, then fill in the successive levels, refining and expanding the original plans until the design is complete. An example of the evolution of a program development from data flow to structure chart, and then to the first level of pseudo-code is shown in Appendix A.

The basic control logic structures and indentation in pseudo- code are used in a carefully controlled way. Pseudo-code uses statements similar to programming statements but they are not bound by formal syntactical rules. Pseudo-code is used to spell out detailed design logic. Because of these similarities the translation from pseudo-code functional statements to the programming language statements should be straightforward.

Meaningful data, procedure, or label names in the form of an action word plus a qualified object are used to identify the purpose or function of the data or processing procedures and program elements.

Program statements implementing control logic structures are indented to show the scope of influence of the structures.

3.6 Efficiency Considerations

A "highly efficient" program that is very difficult to maintain is not really efficient in the context of total cost. Emphasis must also be placed on program maintainability and effectiveness as well as computer efficiency.

3.7 Fine Tuning the Program

Fine tuning a program may become necessary to optimize the use of the CPU. One way to minimize the job's time would be to identify those portions of the program that are most heavily used and concentrate on those few segments. Recode the one-time "called" procedures into in-line code. The short, heavily used loops can be optimized by moving statements that are not directly affected by the loop outside of the loop. Do not use the compiler for data conversions.

Keep the specification of the I/O verbs down to a bare minimum, ideally only one I/O statement per file.

If excessive paging in a virtual storage system is a problem, place procedures that are used together close to each other so their load modules should be placed in the same virtual page.

3.8 References

In addition to this discussion on Structured Programming, a similar discussion may be found in the SDM/Structured SIS manual. And a related discussion may be found in Chapter 23 of the DYL-280 reference manual, standards document 70.05.

4 PROGRAMMING AND TESTING

The State's established programming language standards should be followed during the development of programming code for any program development. Each agency dp coordinator has a copy of the standards for the languages supported at ICSD. By following these standards, the programs within application systems should then be uniform and consistent in structure, style, and content. The project analyst is responsible for establishing common subroutines to standardize the common processes within a system. The programming code should be self-documenting, readable, and easy to modify, maintain, and update by other maintenance programming personnel.

The project system analyst is responsible for developing the system test plan and strategy for system integration. For the continuity and integrity of the system throughout its lifetime, the programmer should develop a permanent set of data that should test all branches in the program. This benchmarking data set should reside in a library of test data sets. This data should be reapplied to the program whenever the program is enhanced or modified.

4.1 Programming Languages

The State's Executive Branch has standardized on COBOL. Programs may be developed in PL/I, FORTRAN, or Basic Assembler Language if there is a need for compatibility with existing software or if the application is better suited for these languages. State supported data base languages are ADABAS and NATURAL on the IBM. On-line IBM-based applications may be created with CICS. The programmer may recommend an application programming language, but the project system analyst is responsible for the final determination of the program's language. All of these languages have a State standards document that the agency dp coordinator has on file.

4.2 Utility Software

Utility software are general-purpose programs to perform commonly executed functions. Some utilities available at ICSD include EasyTrieve Plus, IBM/ISPF, FDR (Fast Dump Restore), PANVALET, and CA-SORT. A listing and description of utility softwares that are available at ICSD may be found in standards document 04.01.

4.3 Programmer Productivity Tools

ICSD supports several application development tools. A productivity software product from Computer Associates named CA-Optimizer can be used to improve COBOL program development and testing time. Training in the use of these products is coordinated through the agency dp coordinator.

4.4 Program Testing

Program testing procedures begins with desk checking of the language code for spelling errors. The second level of testing is the execution of the program with the language compiler to achieve a syntactically correct program. The third level of testing is to check the logic of the program so that the program does what it was intended to do in the simplest and most efficient manner, and that all logical paths are tested out.

4.4.1 Test Data Sets

The State has software products that are capable of creating test data. The reference manual and supporting documentation to use utilities is available in standards attachments – Vendor Materials. The IBM System Utilities manuals can be acquired by the DP Coordinator for each Executive Branch department through their IBM marketing representative.

4.5 Program Documentation

Program documentation for production jobs must be standardized to minimize any need for intervention between the person who developed the program and the person who will be maintaining the program. The program developer must supplement the self-documented program with:

- The system flowchart showing the placement of the program in the system.
- The data set names of the data to be used in the program, the data sets and reports that will be sent out of the program, and the benchmarking test data sets with their DDNAME's.
- A listing of the application defined cataloged procedures in the job stream.
- A listing of JCL control parameters or control statement members stored in a parameter library like "EDPD.PARMLIB."
- A listing of the sample execution run of the program and sample output reports.

5 PROGRAM IMPLEMENTATION

The implementation of a production program must be preceded by conversion procedures developed by the project system analyst. The conversion procedures should include a phase to train the users of the program and steps to migrate the users operations to implement and use the program.

5.1 Source Program Storage

All source code to run on the IBM systems will be stored in PANVALET files. During the testing mode, they will be stored in the data set "EDPD.PANVTEST." During the production mode, they should be stored in the data set "EDPD.PANVPROD." The execution procedures can be found in the "PANVALET User's Guide," standards document 11.04.01.

5.2 Load Module Program Storage

All load modules for test programs on the IBM mainframe systems will be stored on the ICSD system data set "EDPD.LINKLIBT." All load modules for production programs will be stored on the ICSD system data set "EDPD.LINKLIBP."

5.3 Converting from Test to Production

Before the program is turned in as a completed task, the programmer should refer to standards document 07.05 to find out the procedures and documentation that must be accomplished to convert the program from test to production.

5.4 User's Documentation

The recommended contents of the User's Documentation may be found in SDM/Structured Volume VIII, Implementation Phase. In most cases, the project group that developed the application is responsible for developing the user's documentation.

5.5 Operation's Documentation

In most cases, the project group that develops the application is responsible for developing the operation's documentation. The description and contents of the required documents for the "Operation's Documentation" is found in the "ICSD Operations Documentation for Production Jobs," standards document 07.05.

5.6 Scheduling Test Jobs

The programmer is usually responsible for scheduling computer resources, time for test jobs, and for coordinating the program's operational needs with the Production Services Branch Scheduler. But for very large or special resource requirements, the project manager should coordinate the test job scheduling.

5.7 Creating Production Job Streams

Once tested to the satisfaction of the agency, the regular submission of the system's job control and procedures stream should either be transferred from the applications development staff to the user, or transferred from the applications development staff to the Production Services Branch. A cataloged job control production stream must be created.

Before control of any IBM mainframe-based application is transferred from the applications staff, the programmer must transfer all load modules from the test development library, "EDPD.LINKLIBT," to the production library. This production library is usually "EDPD.LINKLIBP," but it may be an agency-assigned partitioned data set.

The programmer will transfer IBM mainframe based application load modules through TSO using ISPF.

- Test to Production - LINKLIBT to LINKLIBP

  Any application system to be handled by the Production Services Control Section will have its load modules stored in the production library defined by the Systems Services Branch which is usually "EDPD.LINKLIBP."

- Releasing EDPD.LINKLIBT Space

  The physical transfer of the application's load module from test to production occurs immediately when the control’s transfer job executes. But the physical delete of the load modules from "EDPD.LINKLIBT" is done only once a day during the system backups which are usually done by Production Services very early the next morning.

APPENDIX A

Structured Program Development

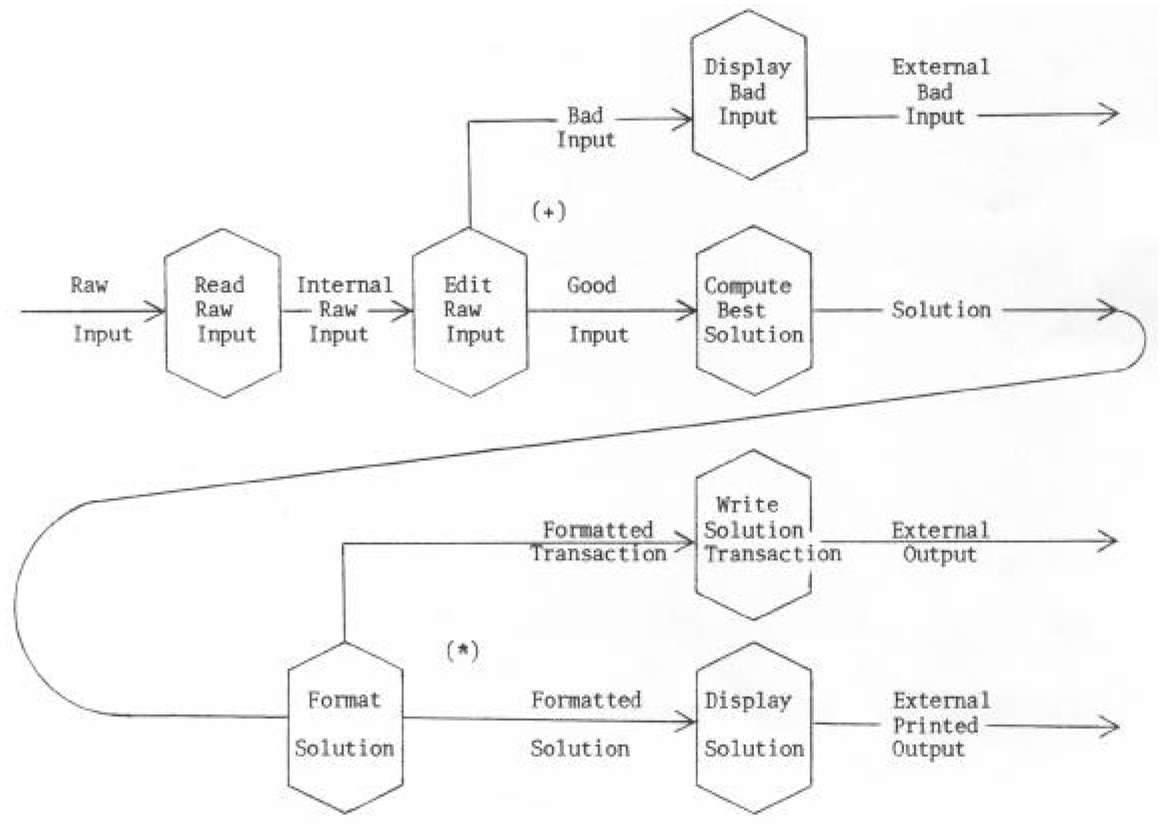
This example illustrates the transitions from a data flow diagram to a hierarchical structured chart and to the initial pseudo-code of the program logic. This pseudo-code is the foundation for the program to be developed. The details for the program comes from expanding the application specifications into detailed procedural steps.

A definition for the special characters on the data flow diagram are:

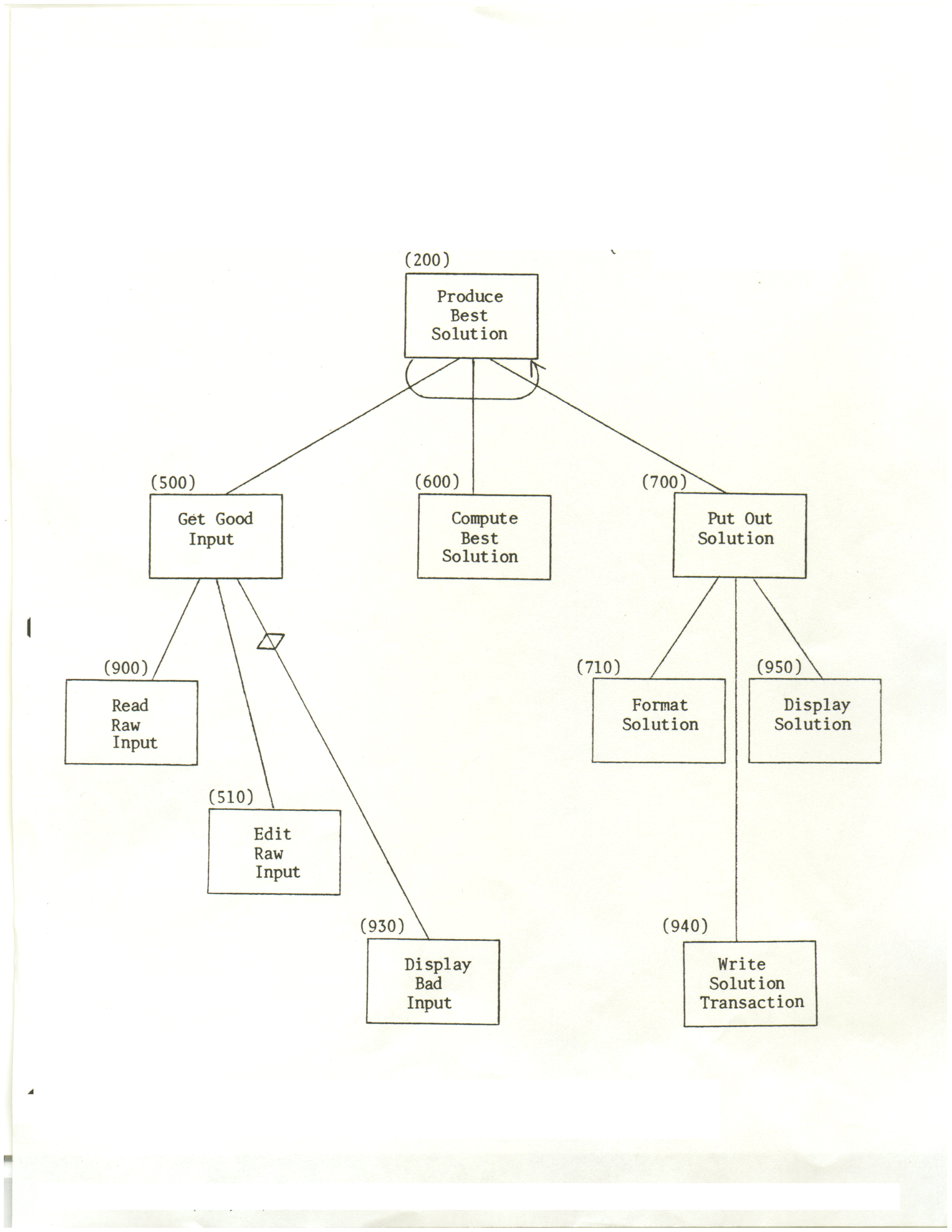
"+" Identifies mutually exclusive processes.

"*" Identifies multiple paths that must be processed.

(+)

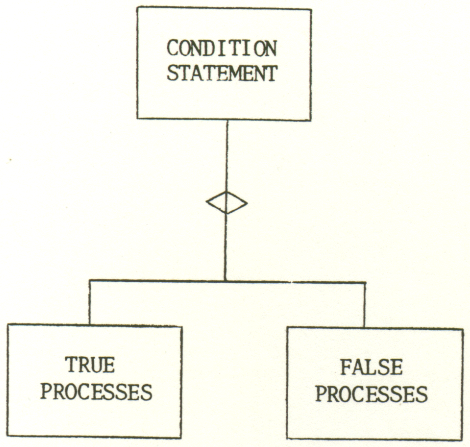


Hypothetical Data Flow Diagram



Hypothetical Program Hierarchical Structure Chart

" ◊ " the diamond identified a decision process. The following structure is used to illustrate an IF-THEN-ELSE process.



" " the looped arrow identifies a group of processes that is repeated in an iteration.

The first level of pseudo-code logic is developed by reading the functional modules on the previous hierarchical structure chart from left to right and top to bottom. Those functional modules had been identified from the data flow diagram.

The next level of the pseudo-code evolution is the systematic expansion of these functional modules with general processing steps identified from the application specifications or with dummy modules if the processes are still being defined. This pseudo-code is converted to a programming language and testing is done after each increment of the functional module expansions is developed.

Pseudo-code logic development is an iterative process with more detailed processing statements specified at the next level of the pseudo-code.

```
ASSUME THERE-IS-MORE-RAW-DATA
REPEAT 200-BEST-SOLUTION
   WHILE THERE-IS-MORE-RAW-DATA

200-BEST-SOLUTION
   PERFORM 500-GET-GOOD-INPUT
   IF THERE-IS-NO-MORE-RAW-DATA
   THEN
               STOP-THE-JOB
   PERFORM 600-COMPUTE-BEST-SOLUTION
   PERFORM 700-PUT-OUT-SOLUTION
200-BEST-SOLUTION-ENDED

500-GET-GOOD-INPUT
   PERFORM 900-READ-RAW-INPUT
   PERFORM 510-EDIT-RAW-INPUT
   IF BAD-RAW-INPUT
   THEN
       PERFORM 930-DISPLAY-BAD-INPUT
500-GET-GOOD-INPUT-ENDED

700-PUT-OUT-SOLUTION
   PERFORM 710-FORMAT-SOLUTION
   PERFORM 940-WRITE-SOLUTION-TRANSACTION
   PERFORM 950-DISPLAY-SOLUTION
700-PUT-OUT-SOLUTION-ENDED
```

Hypothetical Pseudo-Code created from Hypothetical Program Hierarchical Structure Chart by a hypothetical programmer.